Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Dr. Daniel Büscher, Hannes Saffrich
Wintersemester 2019

Universität Freiburg
Institut für Informatik

Übungsblatt 5

Abgabe: Montag, 25.11.2019, 9:00 Uhr morgens

Kommentieren Sie alle Funktionen die Sie auf diesem Übungsblatt implementieren mit *docstrings*.

Aufgabe 5.1 (Konkatenation; Datei: `concatenated.py`; Punkte: 3)

Implementieren Sie eine Funktion `concatenated(xs: list) -> list`, welche eine Liste von Listen als Argument erhält, diese zu einer einzigen Liste zusammenfügt, und die resultierende Liste zurückgibt. Zum Beispiel:

```
>>> concatenated([[1, 2], [], [3], [4, 5]])
[1, 2, 3, 4, 5]
>>> concatenated([])
[]
>>> concatenated([[], [], [], []])
[]
```

Aufgabe 5.2 (Filter; Datei: `overcooked.py`; Punkte: 6)

Gegeben sei eine Liste `recipes` mit Rezepten, wie folgt:

```
recipes = [("Daiquiri", ["Rum", "Limette", "Zucker"]),
           ("Mojito", ["Rum", "Limette", "Zucker", "Minze", "Soda"]),
           ("Whiskey Sour", ["Whiskey", "Zitrone", "Zucker"]),
           ("Tequila Sour", ["Tequila", "Zitrone", "Zucker"]),
           ("Moscow Mule", ["Vodka", "Limette", "Ginger ale"]),
           ("Munich Mule", ["Gin", "Limette", "Ginger ale"]),
           ("Cuba Libre", ["Rum", "Coke"])]
```

Schreiben Sie eine Funktion `mixable(xs: list) -> list`, welche eine Liste `xs` an Zutaten (jede Zutat ist ein String) als Argument erhält und alle Rezepte aus `recipes` zurückgibt, welche mit den gegebenen Zutaten herstellbar sind. Zum Beispiel:

```
>>> mixable(["Rum", "Whiskey", "Limette", "Zucker", "Coke", "Zitrone"])
['Daiquiri', 'Whiskey Sour', 'Cuba Libre']
>>> mixable(["Rum", "Vodka", "Limette", "Zucker", "Ginger ale"])
['Daiquiri', 'Moscow Mule']
```

Aufgabe 5.3 (Binärdarstellung; Datei: `binary.py`; Punkte: 3)

Implementieren Sie eine Funktion `to_binary(x: int, n: int) -> list`, welche die Binärdarstellung der Zahl `x` als Liste mit Länge `n` und in geeigneter Reihenfolge zurückgibt. Falls zur Darstellung der Zahl mehr Stellen nötig sind, sollen trotzdem nur die letzten `n` Stellen zurückgegeben werden. Zum Beispiel:

```
>>> to_binary(33, 8)
[0, 0, 1, 0, 0, 0, 0, 1]
>>> to_binary(458757, 20)
[0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1]
>>> to_binary(458757, 18)
[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1]
```

Aufgabe 5.4 (Primzahlen; Datei: `primes.py`; Punkte: 6)

Primzahlen sind natürliche Zahlen ≥ 2, die durch genau zwei Zahlen teilbar sind: durch 1 und durch sich selbst. Damit ist 2 die kleinste Primzahl, eine größte Primzahl existiert nicht. Implementieren Sie eine Funktion `primes(n: int) -> list`, welche alle Primzahlen kleiner oder gleich n berechnet und diese in aufsteigender Reihenfolge als Liste zurückgibt. Nutzen Sie dazu die folgende Idee: Um zu überprüfen, ob eine Zahl n prim ist, reicht es, diese auf Teilbarkeit durch alle zuvor erzeugten Primzahlen $\leq n$ zu überprüfen. Bereits erzeugte Primzahlen können (und sollten) in einer Liste zwischengespeichert werden. Sie können Ihre Funktion z.B. wie folgt testen:

```
>>> primes(1) == []
True
>>> primes(3) == [2, 3]
True
>>> primes(20) == [2, 3, 5, 7, 11, 13, 17, 19]
True
```

Aufgabe 5.5 (Erfahrungen; Datei: `erfahrungen.txt`; Punkte: 2)

Legen Sie im Unterverzeichnis `sheet05` eine Textdatei `erfahrungen.txt` an. Notieren Sie in dieser Datei kurz Ihre Erfahrungen beim Bearbeiten der Übungsaufgaben (Probleme, Bezug zur Vorlesung, Interessantes, benötigter Zeitaufwand, etc.).